IMPORTING THE IMPORTANT LIBRARIES

```
import pandas as pd
import pickle
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
import xgboost as xgb
import lightgbm as lgb
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv("/content/dataset.csv")
df.head()
```

	name	description	make	model	year	price	engine	cylinders	fuel	mileage	transmission	trim	body	doors	ext
0	2024 Jeep Wagoneer Series II	\n \n Heated Leather Seats, Nav Sy...	Jeep	Wagoneer	2024	74600.0	24V GDI DOHC Twin Turbo	6.0	Gasoline	10.0	8-Speed Automatic	Series II	SUV	4.0	
1	2024 Jeep Grand Cherokee Laredo	AI West is committed to offering every custome...	Jeep	Grand Cherokee	2024	50170.0	OHV	6.0	Gasoline	1.0	8-Speed Automatic	Laredo	SUV	4.0	
2	2024 GMC Yukon XL Denali	NaN	GMC	Yukon XL	2024	96410.0	6.2L V-8 gasoline direct injection, variable v...	8.0	Gasoline	0.0	Automatic	Denali	SUV	4.0	S
3	2023 Dodge Durango Pursuit	White Knuckle Clearcoat 2023 Dodge Durango Pur...	Dodge	Durango	2023	46835.0	16V MPFI OHV	8.0	Gasoline	32.0	8-Speed Automatic	Pursuit	SUV	4.0	V
4	2024 RAM 3500 Laramie	\n \n 2024 Ram 3500 Laramie Billet...	RAM	3500	2024	81663.0	24V DDI OHV Turbo Diesel	6.0	Diesel	10.0	6-Speed Automatic	Laramie	Pickup Truck	4.0	

Data Analysis

```
# Basic information about the dataset
print("\n===== Basic Information =====")
print(df.info())

print("\n===== Statistical Summary =====")
print(df.describe())

print("\n===== Missing Values =====")
missing = df.isnull().sum()
print(missing[missing > 0])

print("\n===== Duplicated Values =====")
print(df.duplicated().sum())
```

```
===== Basic Information =====
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1002 entries, 0 to 1001
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            1002 non-null   object
 1   description     946 non-null    object
 2   make            1002 non-null   object
 3   model           1002 non-null   object
 4   year            1002 non-null   int64
 5   price           979 non-null    float64
 6   engine          1000 non-null   object
 7   cylinders       897 non-null    float64
 8   fuel            995 non-null    object
 9   mileage         968 non-null    float64
 10  transmission    1000 non-null   object
 11  trim            1001 non-null   object
 12  body            999 non-null    object
 13  doors           995 non-null    float64
 14  exterior_color  997 non-null    object
 15  interior_color  964 non-null    object
 16  drivetrain      1002 non-null   object
dtypes: float64(4), int64(1), object(12)
memory usage: 133.2+ KB
None

===== Statistical Summary =====
               year          price   cylinders      mileage       doors
count  1002.000000     979.000000  897.000000   968.000000  995.000000
mean   2023.916168   50202.985700    4.975474    69.033058    3.943719
std       0.298109   18700.392062    1.392526   507.435745    0.274409
min    2023.000000       0.000000    0.000000     0.000000    2.000000
25%    2024.000000   36600.000000    4.000000     4.000000    4.000000
50%    2024.000000   47165.000000    4.000000     8.000000    4.000000
75%    2024.000000   58919.500000    6.000000    13.000000    4.000000
max    2025.000000  195895.000000    8.000000  9711.000000    5.000000

===== Missing Values =====
description        56
price              23
engine              2
cylinders         105
fuel                7
mileage            34
transmission        2
trim                1
body                3
doors               7
exterior_color      5
interior_color     38
dtype: int64

===== Duplicated Values =====
24
```

Handling the missing and the duplicate values

```
df=df.drop_duplicates()
print(f"Removed duplicates. New shape: {df.shape}")

#Fill missing numerical values with median
numerical_cols=['mileage','cylinders','doors']
df[numerical_cols]=df[numerical_cols].fillna(df[numerical_cols].median())

#Fill missing categorical values
text_cols=['description', 'engine', 'fuel', 'transmission','trim', 'body', 'exterior_color', 'interior_color']
df[text_cols]=df[text_cols].fillna('Unknown')

#drop the rows with no price
df=df.dropna(subset=['price'])
print(df.isnull().sum())
```

```
Removed duplicates. New shape: (978, 17)
name              0
description       0
make              0
model             0
year              0
price             0
```

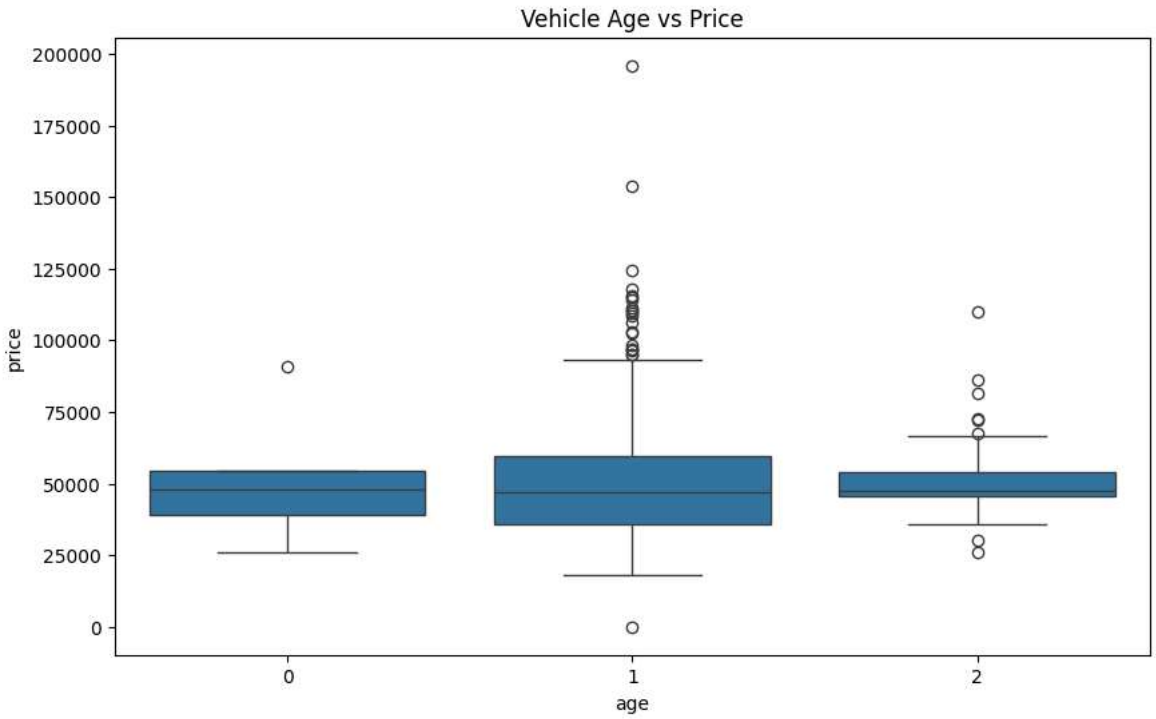
```
engine            0
cylinders         0
fuel              0
mileage           0
transmission      0
trim              0
body              0
doors             0
exterior_color    0
interior_color    0
drivetrain        0
dtype: int64
```

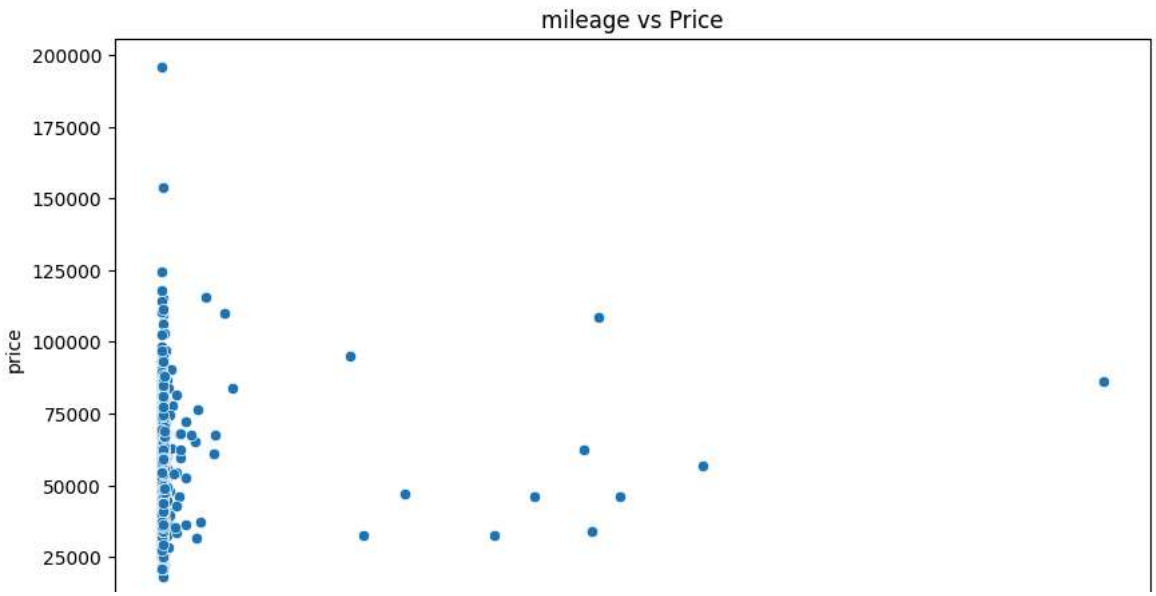
```
# Convert 'year' to 'age' before plotting as year doesn't actually effect the car price but it's age does
if 'year' in df.columns:
    df['age'] = 2025 - df['year']
    df.drop(columns=['year'], inplace=True)  # Remove 'year' since we replaced it with 'age'
    print(df.columns)
#Check for correlation matrix
feat=['age', 'mileage', 'cylinders', 'doors','price']
df[feat].corr()
```

```
Index(['name', 'description', 'make', 'model', 'price', 'engine', 'cylinders',
       'fuel', 'mileage', 'transmission', 'trim', 'body', 'doors',
       'exterior_color', 'interior_color', 'drivetrain', 'age'],
      dtype='object')
```

	age	mileage	cylinders	doors	price
age	1.000000	0.096645	0.270246	-0.077067	-0.004594
mileage	0.096645	1.000000	0.037280	-0.027053	0.077786
cylinders	0.270246	0.037280	1.000000	-0.187729	0.363305
doors	-0.077067	-0.027053	-0.187729	1.000000	-0.071444
price	-0.004594	0.077786	0.363305	-0.071444	1.000000

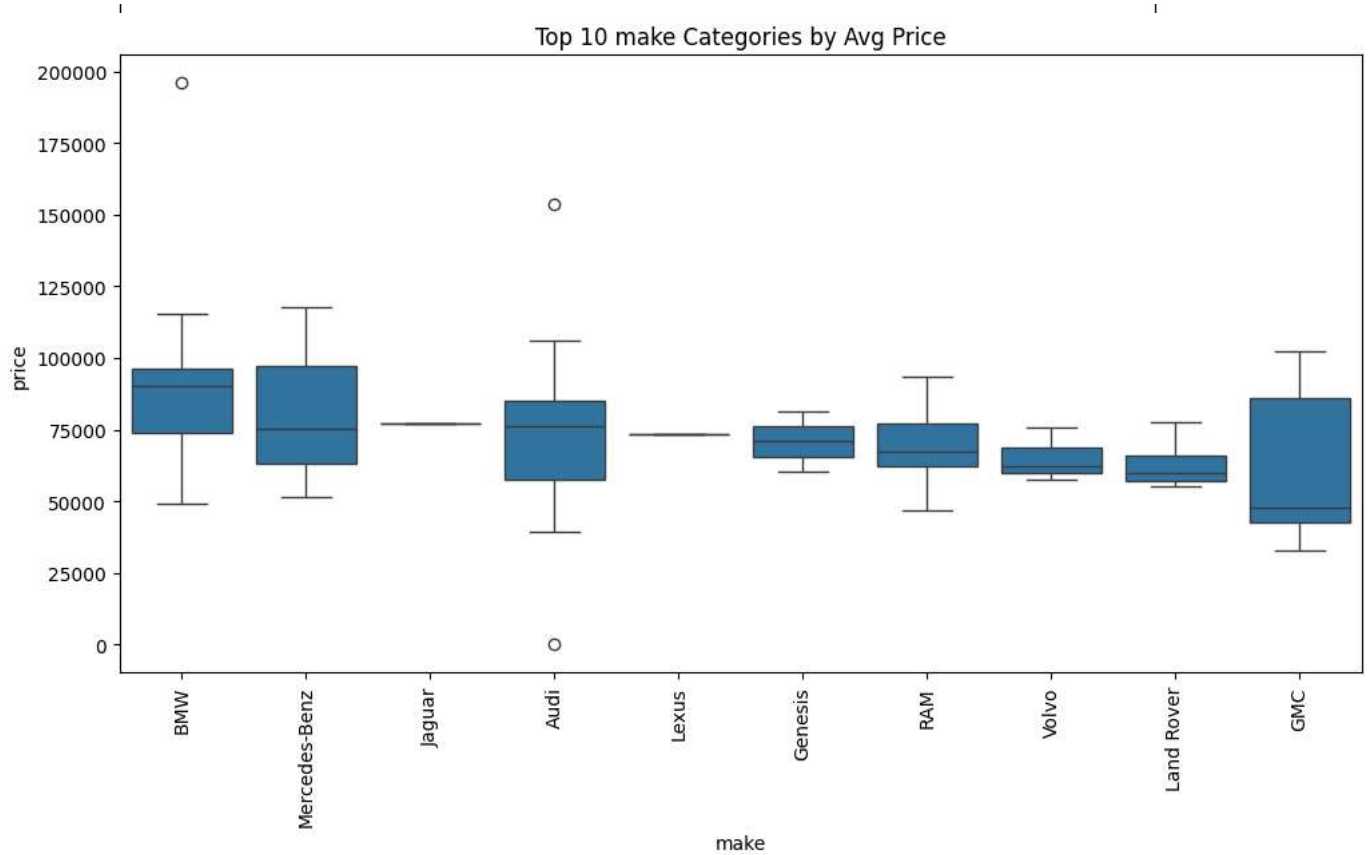
```
# Define numerical features
numerical_features = ['age', 'mileage', 'cylinders', 'doors']
for feature in numerical_features:
    plt.figure(figsize=(10, 6))
    if feature != 'age':
        sns.scatterplot(x=feature, y='price', data=df)
        plt.title(f'{feature} vs Price')
    else:
        sns.boxplot(x='age', y='price', data=df)
        plt.title('Vehicle Age vs Price')
    plt.show()
```

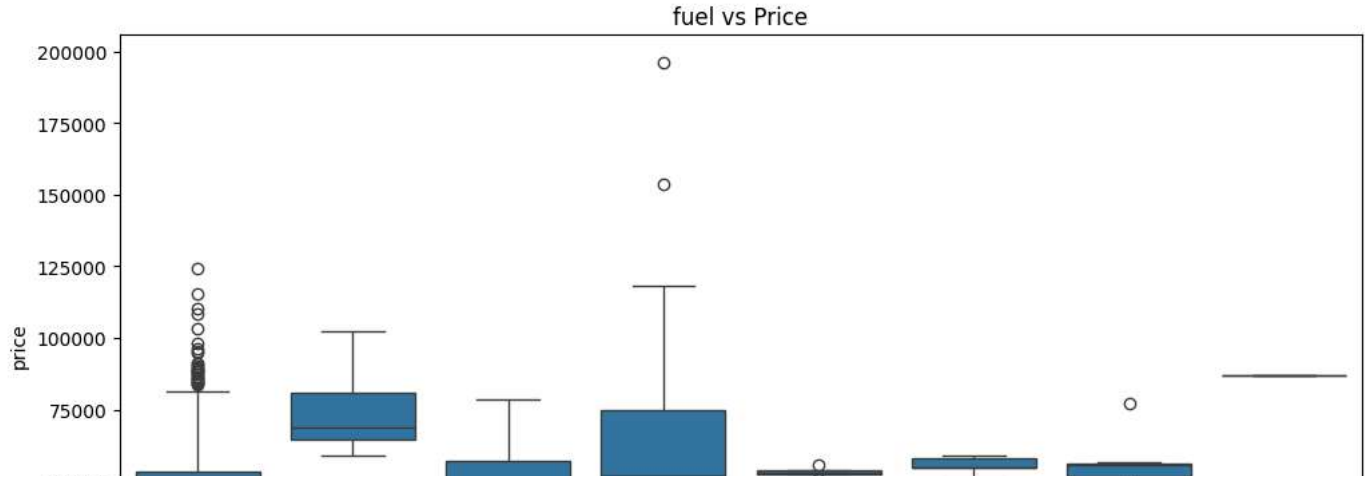





```
# Define categorical features
categorical_features = ['make', 'fuel', 'transmission', 'body', 'drivetrain',
                        'name', 'model', 'engine', 'trim', 'exterior_color', 'interior_color']

# Plot box plots for categorical features
for feature in categorical_features:
    if feature in df.columns:
        unique_values = df[feature].nunique()  # Corrected to count unique values

        # If there are fewer than 15 unique values, plot directly
        if unique_values < 15:
            plt.figure(figsize=(12, 6))
            sns.boxplot(x=feature, y='price', data=df)
            plt.title(f'{feature} vs Price')
            plt.xticks(rotation=90)
            plt.show()

        else:
            # Select top 10 categories by average price
            top_categories = df.groupby(feature)['price'].mean().sort_values(ascending=False).head(10).index.tolist()

            # Filter dataset to include only top categories
            df_filtered = df[df[feature].isin(top_categories)].copy()
```

```
# Convert to categorical for better ordering
df_filtered[feature] = pd.Categorical(
    df_filtered[feature],
    categories=top_categories,  # Enforce sorted order
    ordered=True
)

plt.figure(figsize=(12, 6))
sns.boxplot(x=feature, y='price', data=df_filtered)
plt.title(f'Top 10 {feature} Categories by Avg Price')
plt.xticks(rotation=90)
plt.show()
```





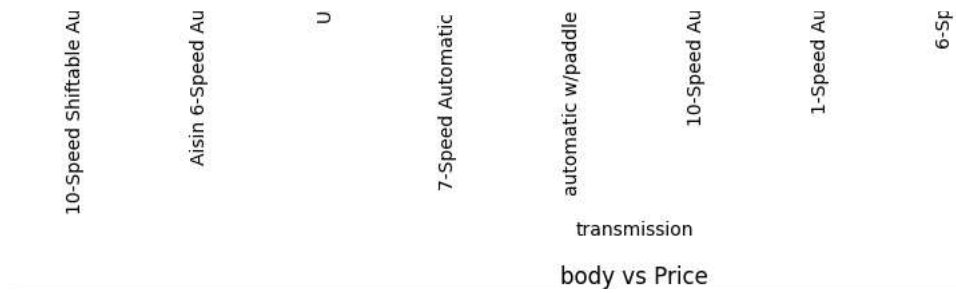

```
numerical_features=['age','cylinders','mileage','doors']
categorical_features=['fuel','transmission','drivetrain','engine','trim','body']
# Create combined categorical feature to reduce cardinality
if {'make', 'model'}.issubset(df.columns):
    df['make_model'] = df['make'] + '_' + df['model']
    categorical_features.append('make_model')
    df.drop(columns=['make', 'model'], inplace=True)
# Reduce high-cardinality categorical feature (e.g., exterior_color)
if 'exterior_color' in df.columns:
    top_colors = df['exterior_color'].value_counts().index[:10]  # Keep top 10 colors
```

```
    df['exterior_color'] = df['exterior_color'].apply(lambda x: x if x in top_colors else 'Other')
    categorical_features.append('exterior_color')
# Drop unnecessary text features
features_to_drop = ['description', 'interior_color','name']
df.drop(columns=[col for col in features_to_drop if col in df.columns], inplace=True)
```

```
Q1 = df[numerical_features].quantile(0.25)  # 25th percentile
Q3 = df[numerical_features].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1  # Interquartile Range

# Define outlier boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Keep only non-outliers
df = df[~((df[numerical_features] < lower_bound) | (df[numerical_features] > upper_bound)).any(axis=1)]
```

```
df.head()
```

	price	engine	cylinders	fuel	mileage	transmission	trim	body	doors	exterior_color	drivetrain	age	make_model
0	74600.0	24V GDI DOHC Twin Turbo	6.0	Gasoline	10.0	8-Speed Automatic	Series II	SUV	4.0	White	Four-wheel Drive	1	Jeep_Wagoneer
1	59150.0	OHV	6.0	Gasoline	1.0	8-Speed Automatic	Laredo	SUV	4.0	Metallic	Four-wheel Drive	1	Jeep_Grand Cherokee
2	96410.0	6.2L V-8 gasoline direct injection, variable v...	8.0	Gasoline	0.0	Automatic	Denali	SUV	4.0	Summit White	Four-wheel Drive	1	GMC_Yukon XL

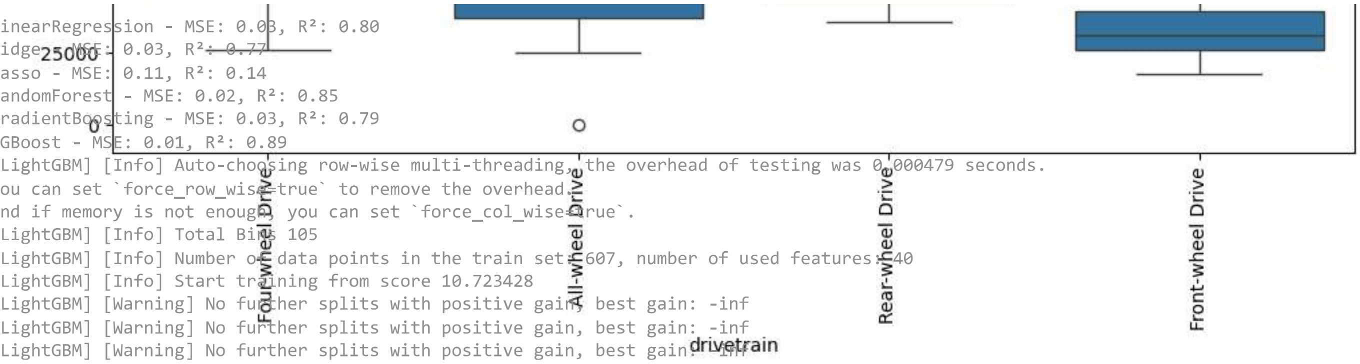



```
transmission_encoder = LabelEncoder()
df['transmission'] = transmission_encoder.fit_transform(df['transmission'].astype(str))

# Save the Encoder
with open('transmission_encoder.pkl', 'wb') as f:
    pickle.dump(transmission_encoder, f)
```

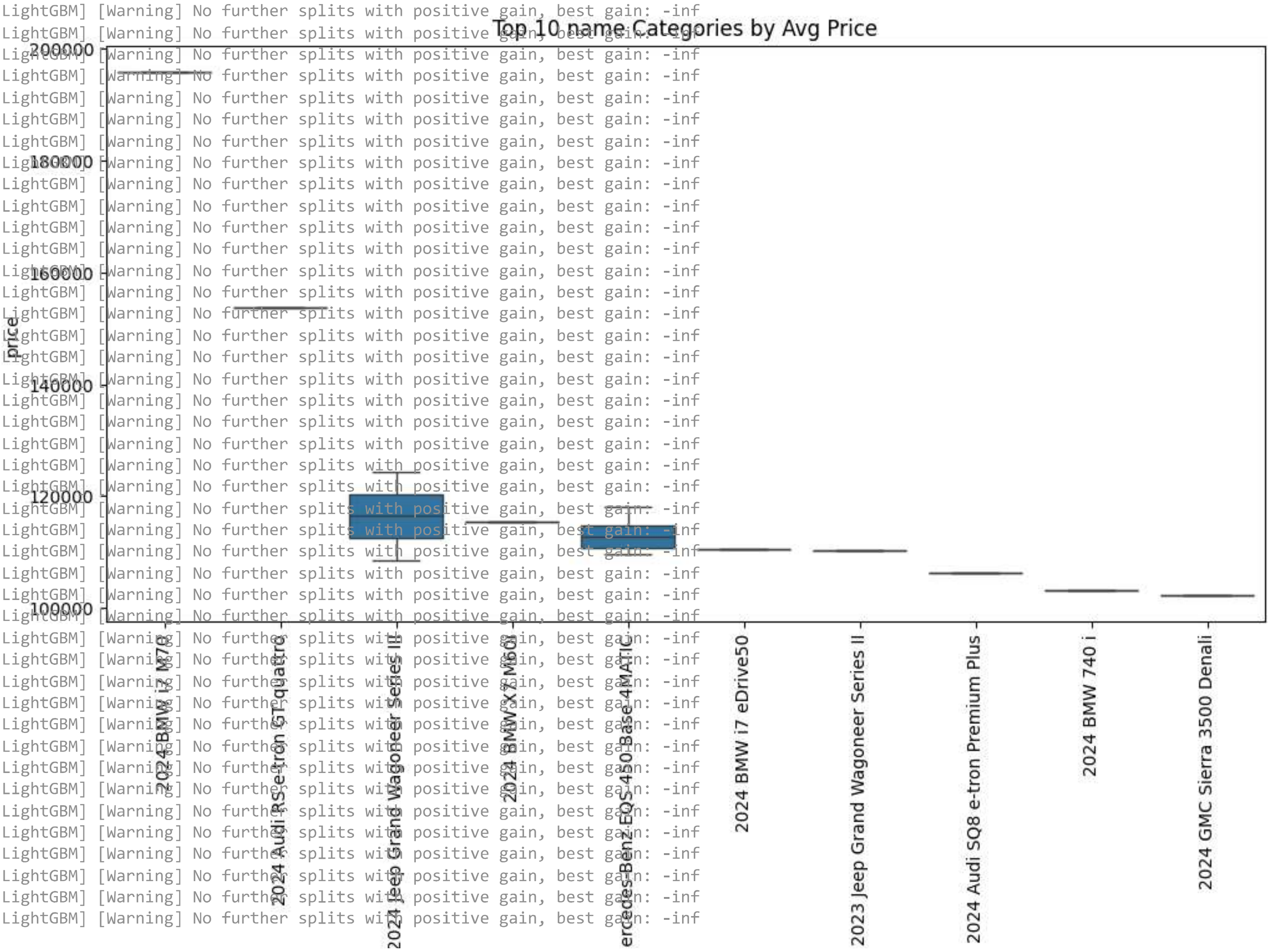
```
# Log-transform price if skewed
if df['price'].skew() > 0.5:
    df['price_log'] = np.log1p(df['price'])
    target = 'price_log'
else:
    target = 'price'
```


```
numerical_features = ['age', 'mileage', 'cylinders','doors']
categorical_features = ['fuel','body', 'transmission', 'drivetrain', 'engine', 'trim', 'make_model', 'exterior_color']

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# Train-test split
X = df[numerical_features + categorical_features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
models = {
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1),
    'RandomForest': RandomForestRegressor(n_estimators=100, random_state=42),
    'GradientBoosting': GradientBoostingRegressor(n_estimators=100, random_state=42),
    'XGBoost': xgb.XGBRegressor(n_estimators=100, random_state=42),
    'LightGBM': lgb.LGBMRegressor(n_estimators=100, random_state=42)
}

for name, model in models.items():
    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"{name} - MSE: {mse:.2f}, R²: {r2:.2f}")
```



```
LinearRegression - MSE: 0.03, R²: 0.80
Ridge - MSE: 0.03, R²: 0.77
Lasso - MSE: 0.11, R²: 0.14
RandomForest - MSE: 0.02, R²: 0.85
GradientBoosting - MSE: 0.03, R²: 0.79
XGBoost - MSE: 0.01, R²: 0.89
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000479 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 105
[LightGBM] [Info] Number of data points in the train set: 607, number of used features: 40
[LightGBM] [Info] Start training from score 10.723428
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```



```
best_model = Pipeline([
    ('preprocessor', preprocessor),
```